0

Introducción Java

```
1.    // fichero Ejemplo1.java
2.    import java.util.Vector;
3.    import java.awt.*;

4.    class Ejemplo1 {
5.        public static void main(String arg[]) throws InterruptedException
6.        {
7.            System.out.println("Comienza main()...");
8.            Circulo c = new Circulo(2.0, 2.0, 4.0);
9.            System.out.println("Radio = " + c.r + " unidades.");
10.           System.out.println("Centro = (" + c.x + "," + c.y + ") unidades.");
11.           Circulo c1 = new Circulo(1.0, 1.0, 2.0);
12.           Circulo c2 = new Circulo(0.0, 0.0, 3.0);
13.           c = c1.elMayor(c2);
14.           System.out.println("El mayor radio es " + c.r + ".");
15.           c = new Circulo();   //  c.r = 0.0;
16.           c = Circulo.elMayor(c1, c2);
17.           System.out.println("El mayor radio es " + c.r + ".");

18.           VentanaCerrable ventana =
19.               new VentanaCerrable("Ventana abierta al mundo...");
20.           ArrayList v = new ArrayList();

21.           CirculoGrafico cg1 = new CirculoGrafico(200, 200, 100, Color.red);
22.           CirculoGrafico cg2 = new CirculoGrafico(300, 200, 100, Color.blue);
23.           RectanguloGrafico rg = new
24.               RectanguloGrafico(50, 50, 450, 350, Color.green);

25.           v.add(cg1);
26.           v.add(cg2);
27.           v.add(rg);

28.           PanelDibujo mipanel = new PanelDibujo(v);
29.           ventana.add(mipanel);
30.           ventana.setSize(500, 400);
31.           ventana.setVisible(true);
32.           System.out.println("Termina main()...");

33.       } // fin de main()

34.   } // fin de class Ejemplo1
```

La sentencia 1 es simplemente un comentario

Las sentencias 2 y 3 importan clases de los **packages** de **Java**. Al hacer esto se puede acceder a la clase **Vector**, por ejemplo, con este nombre, en lugar de utilizar el nombre completo **java.util.Vector**.

La sentencia 4 indica que empieza a definirse la clase **Ejemplo1**. Como en C#, utilizamos la llaves para definir un bloque de código. En **Java** todo son clases: no se puede definir una variable o una función que no pertenezcan a una clase. Una **clase** es una agrupación de **variables miembro** (datos) y **funciones miembro** (métodos) que operan sobre dichos datos y permiten comunicarse con otras clases. Un **objeto** o **instancia** en una variable concreta de una clase, con su propia copia de las variables miembro.

Las líneas 5 a 33 contienen la definición del programa, cuya ejecución comienza siempre por el método **main()**. La palabra **public** indica que esta función puede ser llamada por cualquier otra clase, la palabra **static** indica que es un método de clase, es decir que puede ser utilizado aunque no se haya creado un

objeto de dicha clase. El tipo **void** indica la ausencia de retorno del método. El string **arg[]** se utiliza como parámetros al programa cuando este se invoca.

La sentencia 7 imprime una cadena de caracteres en la salida estándar, que normalmente es la pantalla. Hay que tener en cuenta que **out** es una variable **miembro static** de la clase **System**.

La sentencia 8 es muy propia de **Java**, en ella se crea un objeto de la clase **Circulo**. Primeo se crea la referencia **c** y a continuación se crea el objeto con el operador **new**, que es el **constructor** de clase.

La sentencia 9 vuelve a visualizar un texto por pantalla.

Las sentencias 11 y 12 crean dos objetos nuevos de la clase **Circulo**.

La sentencia 13 (*c=c1.elMayor(c2);*) utiliza el método **elMayor()** de la clase **Circulo**. Este método compara los radios de los dos círculos y devuelve una referencia al círculo que tenga mayor radio. Observar que c1 funciona como un argumento implícito, mientras que c2 es un argumento explícito.

En la línea 14 se imprime el resultado de la comparación anterior.

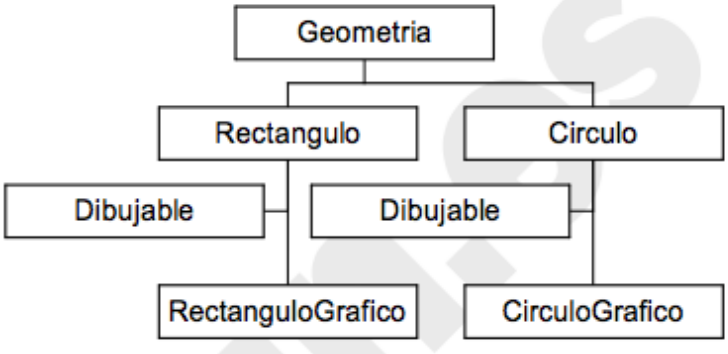
En la línea 15 se crea un nuevo objeto de la clase **Circulo**, en este caso no se pasan argumentos al constructor de la clase, es decir utilizará algún valor por defecto para el centro y radio.

La sentencia 16 (*c=Circulo.elMayor(c1,c2);*) vuelve a utilizar un método llamado **elMayor** para comparar dos círculos. No es el mismo método que el utilizado anteriormente en la línea 13, aunque tenga el mismo nombre. A los métodos con el mismo nombre peo con código distinto se les llama **métodos sobrecargados**. Evidentemente se diferencian por el número y tipo de argumentos. El método de la línea 16 es un método **static** (o de clase), por lo que no requiere de un argumento implícito.

Las sentencias 18-31 tienen que ver con la parte gráfica del ejemplo. Leer documentación sobre **Java**.

SUPER CLASE

Se suele utilizar la nomenclatura super-clase y sub-clase para referirse a la clase padre o hija de una clase determinada.



Vamos a definir en este apartado las clases **Rectangulo** y **Circulo**, que derivarán de otra clase llamada **Geometría**. Como no va a haber nunca objetos de la clase **Geometría**, la definiremos como una **clase abstracta**.

```
1.    // fichero Geometria.java

2.    public abstract class Geometria {
3.        // clase abstracta que no puede tener objetos

4.        public abstract double perimetro();
5.        public abstract double area();

6.    }
```

En primer lugar se declara como **public** para permitir que sea utilizada por cualquier otra clase, y como **abstract** para indicar que no se permite crear objetos de esta clase. Esta clase no tiene variables miembro, peo si declara dos métodos: **perimetro()** y **area()**. Ambos métodos se declaran como **public** para que puedan ser llamados por otras clases y como **abstract** para indicar que no se da ninguna definición de ellos (no tienen código). Es completamente lógico que no se definan en esta clase estos métodos: la forma de calcular un perímetro o un área es completamente distinta en un rectángulo y en un círculo, y por eso estos métodos habrá que definirlos en las clases **Rectangulo** y **Circulo**.

CLASE RECTÁNGULO

La clase rectángulo deriva de Geometría. Esto se indica en la sentencia 2 con la palabra **extends**.

```
1.    // fichero Rectangulo.java

2.    public class Rectangulo extends Geometria {
3.        // definición de variables miembro de la claes
4.        private static int numRectangulos = 0;
5.        protected double x1, y1, x2, y2;

6.        // constructores de la clase
7.        public Rectangulo(double p1x, double p1y, double p2x, double p2y) {
8.            x1 = p1x;
9.            x2 = p2x;
10.           y1 = p1y;
11.           y2 = p2y;
12.           numRectangulos++;
13.       }
14.       public Rectangulo(){ this(0, 0, 1.0, 1.0); }

15.       // definición de métodos
16.       public double perimetro() { return 2.0 * ((x1-x2)+(y1-y2)); }
17.       public double area() { return (x1-x2)*(y1-y2); }

18.   } // fin de la clase Rectangulo
```

Esta clase define cinco variables miembro. En la sentencia 4 se define una variable miembro **static**, que se caracterizan por ser propias de la clase y no del objeto. La variable **numRectangulos** pretende llevar la cuenta en todo momento del número de objetos de esta clase que se han creado. Además está definida como **private**, lo cual quiere decir que solo los métodos miembro de esta clase tienen permiso para utilizarla.

La sentencia 5 define cuatro nuevas variables que representan las coordenadas de dos vértices opuestos del rectángulo. Al declararlas **protected** se indica que sólo esta clase, las que deriven de ella y las clases del propio packaged tienen permiso para utilizarlas.

Las sentencias 7-14 definen los **constructores** de la clase. Los constructores no tienen valor de retorno y su nombre coincide con el de la clase. Los constructores son un ejemplo clásico de métodos sobrecargados. En este caso hay dos, el segundo de ellos no tiene argumentos, por lo que se le reconoce como **constructor por defecto**.

El otro constructor es el **constructor general**, y como vemos recibe 4 argumentos que son asignadas a las variables miembro.

El constructor por defecto llama al constructor general utilizando para ello la palabra reservada **this** seguida del valor de los argumentos.

A continuación se definen los métodos perímetro y área.

CLASE CÍRCULO

La clase rectángulo deriva de Geometría y es muy similar a la clase **Rectangulo**.

```
1.     // fichero Circulo.java

2.     public class Circulo extends Geometria {
3.         static int numCirculos = 0;
4.         public static final double PI=3.14159265358979323846;
5.         public double x, y, r;

6.         public Circulo(double x, double y, double r) {
7.             this.x=x; this.y=y; this.r=r;
8.             numCirculos++;
9.         }

10.        public Circulo(double r) { this(0.0, 0.0, r); }
11.        public Circulo(Circulo c) { this(c.x, c.y, c.r); }
12.        public Circulo() { this(0.0, 0.0, 1.0); }

13.        public double perimetro() { return 2.0 * PI * r; }
14.        public double area() { return PI * r * r; }

15.        // método de objeto para comparar círculos



16.        public Circulo elMayor(Circulo c) {
17.            if (this.r>=c.r) return this; else return c;
18.        }

19.        // método de clase para comparar círculos
20.        public static Circulo elMayor(Circulo c, Circulo d) {
21.            if (c.r>=d.r) return c; else return d;
22.        }

23.    } // fin de la clase Circulo
```

La sentencia4 define una variable **static** pero también **final**. Una variable de este tipo tiene como característica que su valor no puede ser modificado (es una constante).

La sentencia 5 define las variables miembro de la clase, que son las coordenadas del centro y el radio.

La sentencia 6-9 defina el constructor general de la clase. Como los argumentos coinciden con el nombre de las variables miembro, es necesario diferenciarlas. Para ello se vuelve a utilizar la palabra reservada **this**.

Las sentencias 10 a 12 presentan otros tres constructores de la clase.

El primero indica que cuando se pasa un única argumento (el radio) asume por defecto que las coordenadas del centro son (0,0).

En el segundo se recibe un círculo y se hace una copia del mismo.

El tercero sin argumentos asume que las coordenadas del centro son (0,0) y el radio uno.

Las sentencias 13 y 14 definen los métodos perímetro y área, declarados como **abstract** en la clase **Geometría**.

Las sentencias 16-18 definen el método **elMayor()**, que es un método para comparar círculos. Uno de los círculos le llega como argumento implícito (el mismo) y el otro como explícito. Compara el radio del objeto implícito **this.r** con el del objeto pasado como parámetro **c.r**.

El otro método elMayor() está definido como static, y por lo tanto no tiene argumento implícito. Recibe dos objetos círculo y compara su radio.

CONCEPTO DE INTERFAZ

El concepto de **interfaz** es muy importante en **Java**. A diferencia de *C++*,**Java** no permite **herencia múltiple**, esto es, no permite que una clase derive de dos clases distintas. Las interfaces de Java constituyen una alternativa a la herencia múltiple.

Una **interface** es un conjunto de declaraciones de métodos (sin implementación). Cuando una clase implementa una determinada **interface**, se compromete a definir todos los métodos de la misma (es como una clase **abstract** con todos sus métodos **abstract**).

```
1.  // fichero Dibujable.java
2.  import java.awt.Graphics;
3.  public interface Dibujable {
4.      public void setPosicion(double x, double y);
5.      public void dibujar(Graphics dw);
6.  }
```